Suitability of Agile Methodology in Globally Distributed Software Development: A Case Study

Suleyman Akbas

Helsinki May 22, 2019

Master's Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Suleyman Akbas

Työn nimi — Arbetets titel — Title

Suitability of Agile Methodology in Globally Distributed Software Development: A Case Study

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	May 22, 2019	59 pages + 3 appendices

Tiivistelmä — Referat — Abstract

This thesis investigates the suitability of agile methodology in distributed software development. This is done by first identifying the challenges of distributed software development which are, by the reviewed literature, communication and collaboration, decrease in teamness feeling, architectural and technical challenges, and decreased visibility for the project status. Then, the thesis presents the agile methodology with its two methods, namely Scrum and Extreme Programming (XP). Thirdly, the benefits and the challenges of applying the agile methodology in distributed software development are determined.

Findings from literature are tested versus a case study which was done in a globally distributed software development team who had worked on an important project in a multinational private software company. The data collection methods were the participant-observation done by the author as a part of the team, author's notes on the critical events, and also the semi-structured interviews done with the team members from different roles and different teams.

Empirical results show that agile methodology, more specifically Scrum and XP, helps with many aspects in distributed software development, which include increased communication and collaboration, improved visibility for the project status, and also increased the sense of trust within the team. It is also discovered that agile methodology helps with onboarding new people to the team. Furthermore, limited documentation in agile methodology and virtual pair programming do not affect the distributed teams negatively according to empirical evidence. Finally, empirical data also shows that applying agile methodology in distributed software development has some challenges such as the number of meetings.

Empirical results show resemblance with the reviewed literature in many parts such as increased communication and collaboration as a benefit of distributed agile software development. However, there are also some aspects contradicting the reviewed literature. For example, limited documentation appears as a challenge of distributed agile development in the reviewed literature, whereas it did not seem to be a challenge in the empirical case.

Furthermore, this study can be extended by observing other empirical cases, notably failed projects, not only in software development but also in other fields.

ACM Computing Classification System (CCS): D.2.9 Software engineering - Management

Avainsanat — Nyckelord — Keywords
distributed software development, agile software development

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

In today's world, distributing software development into several locations is an everyday reality because of its advantages such as reaching to new talents and less tangible cost [Mil08] [EN01]. Besides the advantages, there are many challenges of distributed software development which can be found in the literature including but not limited to communication, coordination of work, and designing a suitable architecture [SD10] [Mil08].

According to empirical analysis in literature, distributing the development into several locations makes a negative impact on the overall software quality because of the reduced productivity in the development. However, it is claimed that these adverse effects can be reduced if there is a well-designed software development process [RB07]. In this manner, agile methodology comes into play.

Agile methodology is a group of software development methods that are mostly used in problems or projects to satisfy changes in requirements with iterative and incremental developments [KB12]. Agile methodology brings some practices such as continuous integration and constant communication, which can be beneficial for distributed software development teams [VM08]. However, there is also some research showing that agile methodology brings some challenges to distributed software development [SD10]. Therefore, this brings up the question: how does agile methodology fit in distributed software development? This question can be answered by identifying the challenges of distributed software development, and figuring out if distributed software development can get any benefit or drawback from applying agile methodology.

This thesis focuses on an empirical case study to understand the lessons learned from experienced professionals working in agile software development within a globally distributed team, also called global software development (GSD). The case was analyzed through conducting interviews with experienced people from an international software company. Finally, the results were compared to the literature in order to discuss and come up with an answer to the question if agile methodology was perceived as suitable for distributed software development by the team members. To achieve answering this, the challenges of distributed software development, and the benefits and drawbacks of applying agile methodology in distributed software

development were discussed.

1.2 Research Problem

The research problem of the thesis is:

- How does agile methodology fit in to overcome challenges of globally distributed software development?

The main question is divided into three research questions (RQ):

*RQ*1 What are the challenges of distributed software development?

*RQ*2 How can distributed software development benefit from agile methodology?

*RQ*3 Does agile methodology bring any challenges to distributed software development?

1.3 Scope of Research

The thesis primarily focuses on the challenges of distributed software development and how agile methodology fits in to overcome some of these challenges. The author conducted interviews with several developers, product managers, software architects, and people managers, explicitly working in an ongoing agile software development project within a globally distributed team. Analyzing their answers and comparing them to the literature give the insight to answer the research questions.

It is also discussed if there are still some open problems that cannot be addressed using agile methodology. Moreover, it is investigated if applying agile methodology in a distributed team brings some challenges to the subject. The thesis also touches some project management points, but the main focus is in software development.

Is there a better software development methodology than agile methodology for distributed software development teams? This is not discussed in this thesis. A more comprehensive study can be conducted on this to compare different methodologies to find which one fits better.

Although it is briefly mentioned, the advantages and disadvantages of distributing work into globally distributed teams are not examined. This can be a topic for other research.

1.4 Structure of Thesis

The thesis goes over the literature in Sections 2 to answer the research question 1-3, which are to define the challenges of distributed software development, the practices of the agile methodology that can be beneficial for distributed development, and the challenges agile methodology might bring. Section 3 shows the context of the case and the research methodology. In Section 4, the empirical case study is compared to the findings from the literature to demonstrate if agile methodology was fitting for this case of distributed development as this is the research problem of the thesis. Section 5 discusses these results, and finally, Section 6 concludes the thesis.

2 Agile Methodology in Distributed Software Development

This section explains the concepts around this thesis through examining the literature. It goes over distributed software development, agile methodology, and finally distributed agile software development.

The reviewed literature was found by searching "Distributed Agile Software Development" keywords in Google Scholar and then applying the "snowballing" technique to determine the relevant literature for this study [Woh14].

2.1 Distributed Software Development

With software being more and more important every day for the companies to become successful in their fields, lots of them have begun utilizing remote teams to create software [SD10] [HCAF06].

This is mostly because the companies want to take advantage of the lower costs and to reach talents from all over the world. Also, there are benefits related to business such as being close to the customers and seizing merging or acquisition opportunities. In this way, as a whole, software development has been forming into a multisite, globally distributed process. This affects all the processes in software development from how a product is designed to how it is delivered to the customers [SD10] [HCAF06] [Her07] [SCS06].

In many aspects, distributed software development is different from centralized development. It leads to distributed teams, time zone differences, cultural differences, etc. [SD10].

2.1.1 Global Software Development (GSD)

Geographical distribution of software development can be in two ways. If the company has the headquarters and the development offices in the same country, this is called onshore distribution or distributed software development (DSD). On the other hand, it is called offshore distribution or global software development (GSD) when some parts of the development are done in different countries [SD10] [PHaOH$^+$10]. Especially GSD is getting more popular among the companies because of its benefits such as lower labor cost in some countries. Even though DSD and GSD both

have drawbacks of which some are explained below, globally distributed software development can be more challenging because of the other factors coming into play such as limited face-to-face communication and time zone differences [PHaOH+10].

2.1.2 Benefits of Distributed Software Development

Although it can be preferable by the teams to be as close as possible because of the challenges distributed software development brings which are explained later in this section, certain factors drive the companies to decide to distribute the teams. The list below presents a few of these reasons.

Cost: Even though it is not easy to count up pros and cons for distributing the teams directly or through outsourcing, since the actual cost is usually less than operating a local branch, companies decide on distributing the work [Mil08]. Especially the lower labor costs in some countries attract the companies to open up new development branches in those countries [PHaOH+10].

Market Reach: Companies seek the needed competence in the new markets they entered. The way to achieve this is usually by purchasing the local companies or opening their subsidiaries in these new areas [Mil08]. Also, opening up a branch and creating new jobs for the local people could be a winning strategy to bind new customers from that area [EN01].

Access to New Talents: As the companies grow, it gets more difficult for them to find all the required talents in only one area. Furthermore, the cost of transferring a new employee to a specific location can be a real burden and not possible all the time considering the relocation costs and the visa availability [Mil08]. Moreover, it might be much more expensive to hire people with specific needed skills in the current location than finding someone from a cheaper country [EN01]. That is why companies tend to open up new branches to reach new talents.

Increasing the Development Speed: Increasing the development speed can also be a reason to distribute the development by creating new development offices in other countries which will work in parallel to accelerate the development. Creating a new office in another country can be useful, particularly if it allows finding skilled

people quicker than the host country /citemaria2010. That is why this is also closely related to the previous benefit, access to new talents.

2.1.3 Challenges of Distributed Software Development

The companies should be cautious when deciding on distributing the team.

Although it brings some advantages, global software development has many drawbacks such as more difficult communication and coordination, less trust within the team, and decreased visibility of the project status [Mil08] [HFÅC06].

Communication and Collaboration: Communication is of great importance in software development. It is claimed that the lack of communication generates most of the problems associated with software development in general. It is either a poor communication between the developers, between developers and the users or between architects and developers. Distributed software development makes it worse [Par06]. It has been found that development in the distributed context takes around 2.5 times longer than co-located development due to the problems with communication [HM03]. Being distant gives rise to a decrease in the amount of communication, most importantly, between the colleagues. A study states "casual conversations with the team room, in hallways, and other shared spaces make a significant contribution to the team's collective understanding. Remote team members miss these, and consequently, their understanding suffers" [Mil08]. Because of the decrease in quality of the communication in distributed teams, another study suggests conducting the important meetings such as architectural and important milestone decision meetings onsite instead of virtual meetings [EN01]. These show that communication in distributed development is a real challenge.

Time zone differences also supplement extra challenges to the problem [SD10]. Even though in a well-coordinated case, time zone differences can be advantageous to achieve 24 hours working time, this requires a minimum amount of interdependence between different sites [OO00]. In a study, engineering team members working in a distributed project complain about being frustrated while trying to follow up discussions held in the other offices in a different time zone after they leave the office. This sometimes even requires them to be contacted outside working hours, which later on causes dissatisfaction among these employees [HFÅC06].

Another critical point is the difficulties in understanding when the spoken language

and culture are different between the sites. Distributed teams rarely speak the same language natively or share the same cultural understanding, which may lead to ineffective communication and even misunderstandings. This can be a reason for a significant decline in the performance of the teams [Her07] [OO00].

Teamness: While distributing the team has the advantage of reaching new talents from different areas of the world, it is challenging to keep a "teamness" feeling among the peers. Since developers usually do not have the chance to travel to meet their colleagues from other locations, they do not quickly develop trust and a sense of belonging within the team. This can cause discontent and make it difficult to manage these people [HFÅC06].

Architectural and Technical Challenges: Since it is not trivial to distribute the work among distributed teams, the architecture of the software can negatively be affected by this. Conway's Law says that "organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations" [con]. That is, instead of coming up with the best architecture for the software, the natural approach pushes the architects to design the software in a way that can easily be distributed to the distant teams [Mil08]. Confirming this, a study found out that a team of engineers trying to design a software stopped using technologies to continue working remotely. Instead, they reorganized the work over time in such a way that no remote teams were in need to closely working with one another [OO00].

Moreover, the teams end up having many artifacts based on the distribution, which need to be effectively bundled into one software. Achieving this well is not straightforward [Mil08].

Decreased Visibility of the Project Status: Although the visibility of the project status may be missing also in the co-located software development, distributing the development into different locations make it worse. Primarily the project managers, as the people who need to evaluate the progress of the project, have difficulties understanding the overall status [PDJ09].

2.2 Agile Methodology

Agile development methodologies can be defined as a group of quick and light software development techniques designed to be flexible with the alterations [SD10]. They are proven methods to boost team productivity five to ten times for collocated teams [Coc02].

With the help of these software development approaches, as agile methodologies, it is now easier to adapt to the changing requirements from customers. The central philosophy behind it is to provide the customer with working software at the end of every short iteration so that the customer can review and give feedback. This way, the software is updated in each iteration according to the feedback from the customer. As it gets costly over time to change something in software, being adapted to the change, agile methods provide cost saving as well [MH09].

Providing periodic working releases is not a trivial task but requires strong engineering teams who can work collaboratively with ever-changing requirements [SS08].

Two agile development methods are well-known among many others, which are Scrum and Extreme Programming (XP). While Scrum has a great emphasis on project management, XP is mostly related to the actual implementation of the software [HFÅC06]. Both explained later in this section.

2.2.1 Agile Values & Principles

Agile manifesto defines four values for agile development: "individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, responding to change over following a plan" [agi01]. These are important to know as they form the fundamental understanding behind every agile development method.

The agile manifesto also lists twelve principles which are lightening the road for the ones who want to utilize agile development. Some of them are as follows [agi01]:

1. Satisfying customer by continuously delivering valuable, working software
2. Adapting well to the changing requirements in favor of customer satisfaction
3. Preferring conversations in-person as it is the most efficient way
4. Making developers and business people work together

5. Embracing simplicity
6. Enabling the team to improve themselves based on their reflections

Following these principles, agile software development helps the development to get faster by eliminating non-value adding time-consuming practices while keeping the customer involved for quick feedback for the direction [PP11].

Although the principles stay the same over time, their applications can change and evolve based on the case. The critical aspect is understanding and absorbing the principles so that they can be adapted effectively with the highest gain [PP11].

2.2.2 Scrum

Scrum is a lightweight framework helping to cope with complex problems in product management in order to keep the focus on continuously delivering a high-quality product [Sch04].

Scrum was actually in use even before the Agile Manifesto introduced the agile methodology. Since it had the same underlying principles and purposes such as simplifying project processes and putting less emphasis on documentation, it became a part of agile methodology [MH09] [Sch04].

As an addition to all other agile practices, Scrum brings the concept of backlogs where the user stories are listed with their complexity size that the team estimated upon [Sch04].

A team of Scrum is composed of a Product Owner, a development team, and a Scrum Master. Product Owner is responsible for bringing up the requirements and creating user stories for the team. He is in close relations with the business side. Development team usually consists of the developers, testers, business analysts, architects, etc. who are in charge of choosing the ways, techniques of the solutions and responsible for implementing them [CWP08]. Scrum Master is the person who identifies blockers for the team and helps to solve them. Scrum Master also guides the team not to lose focus on the goals [Sch04].

Scrum Practices

Key practices of Scrum are explained in this section for clarification.

Daily Scrum: Daily Scrum is the status update meeting held every day stating at the same time and lasts around 15 minutes. During this meeting, each member of the team shares their status and denotes any blockers if exist so that the team can plan the next 24 hours and also evaluate their progress for achieving the sprint goal [SS17].

Scrum of Scrums: Scrum of Scrums is the status update meeting among the teams. It can be held once or multiple times per week and the anticipated duration is around half an hour. One member from each team attends to this meeting where they briefly summarize what they have done so far during the week, what they plan to do, and if there are any blockers in their way or they will put any blockers for the other teams [PHaOH$^+$10].

The Sprint: A release is divided into several iterations; each called a sprint. Every sprint should have its own goals and expectations, more precisely one specific feature, that the team must focus on. A sprint can take from one to four weeks. The team decides what to implement for an iteration based on the priorities determined in a planning session [CWP08].

Sprint Planning: In the very beginning of the sprint, a sprint planning meeting takes place where the user stories are prioritized, and some of them are chosen to be implemented [SS08]. All the team members collaborate in this meeting to come up with a concrete plan for what and how to implement a deliverable during this sprint [SS17].

Sprint Review: At the end of each sprint, a sprint review meeting takes place where the team members demonstrate to the stakeholders what was done during the sprint. After receiving feedback from the stakeholders, the team works collaboratively with them to adopt the product backlog in a way to capture the best possible value for the next sprint [SS17].

Sprint Retrospective: Sprint retrospective meeting is also held at the end of every sprint after the sprint review. In this meeting, the team talks about what went well, what did not go well for the sprint, and what they can improve for the

future. It is a chance for the team to check how they do and improve based on the discussing taking place during this meeting [SS17] [PHaOH+10].

Product Backlog: Requirements of the product are collected in an ordered list, called product backlog, in the form of user stories explaining what the end user needs to have in a few sentences [CWP08]. It is an actively evolving list of items based on the current needs of the product [SS17].

Sprint Backlog: Sprint backlog includes the list of items from product backlog which the team decided to achieve during the sprint [SS08]. Like product backlog, sprint backlog also evolves throughout the sprint in order to allow the team to achieve the spring goal [SS17].

2.2.3 Extreme Programming

Extreme Programming (XP) is another lightweight agile methodology aiming to improve software development in terms of quality and responsiveness through improving communication, the feedback cycle, simplicity, and courage. [BA04]. Unline Scrum aiming to improve the project management related practices, XP is particularly interested in the software development practices [HFÅC06].

Extreme Programming Practices

Extreme Programming offers useful practices to improve software development, some of which are explained below.

Pair Programming: Two programmers are working together, collaborating on the same design, algorithm, code or test using one screen [BA04].

Continous Integration: The code being developed is integrated with the rest of the system within a couple of hours. During this process, the system should be built from scratch, and all the integration tests should pass [BA04].

Test-Driven Development: Before writing the actual code, the unit tests are written to keep the development focused by making it clear what the current aim is

and what to do next [BA04].

Coding Standards: Coding standards are the rules determined by the team to standardize the code written. In the end, it should not be possible to understand who has written by looking at any piece of code. This way, the developers can work on any part of the system without any difficulties [BA04].

This section aimed to explain what agile methodology is and two of its most commonly used methods, called Scrum and Extreme Programming (XP). Also, the Scrum and XP practices are presented to make it more clear what it means to apply these methodologies.

2.2.4 Large-Scale Agile

Agile methodology can also be scaled to match the needs of a scaled project. Although only a couple of teams working on the same project can refer to a scaled project, it is usually meant many more teams working on the same project (e.g. one hundred 8 person teams). This requires more practices such as daily Scrum of Scrums where each team is represented by someone to align the work with all the teams [Sch04]. Also, the additional forms of the teams, particularly task forces -people from several teams temporarily joining the forces to overcome a specific problem-, can help with tailoring the needs of the large-scale agile projects [RMN16].

A study in the literature formed a model to gain the same benefits of agile development for the large scale distributed projects. For a transition from collocated agile development to the distributed one, they come up with four different stages [SS08].

Evaluation: In this phase, a project is evaluated for its fit for agile development and distribution. It is graded for the extension of benefits which can be gained by applying agile methodology. An example criterion can be how useful it would be for the project if documentation is reduced. The next step after assuring agile methodology would be a good fit is deciding how distributable the project is. To illustrate, legal requirements for keeping the customer data can limit the locations of the work to a particular area or even to only one location making distribution impossible [SS08].

Inception: After the evaluation phase is complete, the inception phase takes place in which all the teams are composed and equipped with necessary tooling and training. Some of the team members as the representatives from their locations gather in one central location and perform activities together like planning the tasks for their core teams. In this stage, the important thing is that representatives from remote teams get familiarized with each other [SS08].

Transition: In this phase, representatives return to their core teams and go through a sprint together with the team. Teams already start working on specific parts of the project. They begin capturing the ownership of those parts and improve how they organize. Also, there will be interactions between some of the remote teams, and the representatives will play a critical role to make this efficient [SS08].

Steady State: In this last stage, the teams capture full ownership for their parts, and they become more accustomed to communicating with the teams from other locations. Once this is achieved, distributed large-scale agile development will be in track, and there will be no need to keep a big team in the central location [SS08].

This section provides a brief look at large-scale agile and a sample model to make the transition from co-located to distributed agile development. The empirical case analyzed in this thesis does not apply any large-scale agile practices, though it can be regarded as a scaled project as it has more than one team working on the same project. In the next section, the benefits of applying the agile methodology in distributed software development are identified.

2.3 Benefits of Applying Agile Methodology in Distributed Software Development

As the software industry calls for high-quality programs with low cost, having both distributed and agile software development can be worth evaluating for the companies to achieve this. Although agile practices require the team to be in the same room, this is usually not the case as many teams are distributed over the world. That is why agile methodologies need to be extended to include globally distributed software development as well [Mil08] [SD10].

In a survey conducted by VersionOne, 79% of respondents said they had some teams

utilizing the agile methodology in distributed software development [ver18].

Since agile methodology requires constant communication, many studies show that distributed development can benefit from agile practices in order to cope with the difficulties such as less collaboration within the team and the decrease in the trust [SD10] [PHaOH+10]. A statistical analysis done in 2008 has proved that using agile methodology in distributed software development projects improves productivity and project timeline without any adverse effect on the overall effort and quality of the product [SS08].

The same study also claims that applying Extreme Programming practices was beneficial in terms of the technical aspect and applying Scrum practices was useful for planning and tracking in the distributed software development [HFÅC06].

2.3.1 Communication and Collaboration

Primarily daily Scrums being held every day as one of the Scrum practices helps to encourage communication in distributed teams. It even fosters off-line communication right after daily Scrum to discuss more on the details of the tasks [PHaOH+10].

Also, the other Scrum meetings such as planning and retrospective increase the communication and the collaboration between the team members in the distributed software development. These communication-heavy practices allow all the people from different backgrounds, different cultures to talk to each other freely, which breaks the mental barriers between them. [HFÅC06].

A study also confirms this by stating that utilizing all the agile practices is the best solution they ever encountered against poor communication in the distributed context [Sim06].

In a previously done case study, it is discovered that the team members find an XP practice pair programming beneficial to increase the collaboration between the people working in different locations. It is figured out that even though there is a delay in communication from time to time, people are more inclined to spend much more time with their remote peers [HFÅC06].

2.3.2 Teamness

Beside other Scrum practices, mainly retrospective meetings help the people working in distributed development to identify themselves as a part of the team together with

other members [PHaOH+10].

Also, a virtually shared Scrum board showing the current status of the team gives the possibility for everyone to be involved and influence the work being done. This significantly increases the overall teamness feeling, being a part of the same team, for everyone in the distributed development [HFÅC06].

Furthermore, as the main characteristic of agile development, constant communication helps to build trust within the organization composed of people from all over the world. This even improves the motivation of the teams [SD10]

2.3.3 Visibility of the Project Status

Running sprints in Scrum enhances the visibility of the current status for the teams working in a distributed setting. They have a clear deadline and goals which help everyone in the distributed development to get a feeling of what has been done so far and what is the next step [PHaOH+10].

Continuous iterations as a practice of XP makes it easier to detect the issues, and also makes the current status more visible [EN01]. It brings the visibility needed notably by the project manager and the customers so that they can measure the overall progress of the project. This directly challenges one of the fundamental problems in distributed software development, decreased the visibility of the project status [PDJ09].

2.3.4 Integration of New Code

As a practice brought by Extreme Programming, continuous integration helps to reduce the integration problems the distributed teams suffer at the end of each iteration while integrating the part they implemented with the rest of the software. Thanks to the continuous integration, the new code is integrated into the rest of the software within hours after making sure that all the integration tests are green [PDJ09] [BA04].

2.3.5 Faster Time to Market and Responsiveness

A study shows that working incrementally towards a stable build significantly decreases the project cycle time, so this can be a significant factor for success in distributed software development [EN01].

Considering all the benefits brought by agile methodology such as responsiveness to change, applying it in globally distributed teams can be rewarding especially in terms of agility and performance [SD10].

Agile methodology brings numerous benefits to distributed software development. Some of which are listed in this section, and they can be extended with further research.

2.4 Challenges of Distributed Agile Software Development

Although agile methodology brings many benefits to distributed software development as identified in the previous section, some disadvantages come into play [SD10].

2.4.1 Documentation

Agile methodologies give less value to the documentation and put more emphasis on communication instead. However, when the teams are distant from each other, it could be more difficult trying to keep up with the information only through lacking conversation. Therefore, the teams might need digging more into details over some documentation [SD10]. From his more than five years working experience with distributed teams, Simons says "pure agile development [in distributed software development] with little or no documentation beyond code is impractical and inefficient". Instead of eliminating documentation, they come up with lightweight documentation expressing only needed and practical information, especially for distributed teams [Sim06].

2.4.2 Meeting Times in Different Time Zones

Since agile methodology requires extensive collaboration and communication within the teams and with the stakeholders, falling into different time zones brings up a real challenge. To fully utilize agile, distant teams must align for the working time, which might not be convenient for everyone [SD10].

2.4.3 Pair Programming

Sitting together in pairs and working on programming a piece of the solution is one of the standard practices of XP. Since this is not possible for distributed teams, they would lack this essential feature [SD10]. On the one hand, it is claimed that remote pair programming can also be feasible, but only if remote pairs can somehow realize the properties he lists for successful pair programming, some of which are shared goals and plans, and effective communication. Although it is not easy to achieve the same level of efficiency with co-located pair programming, with the help of cross-workspace tools (e.g., visual and audio tools), remote pair-programming is in some way achievable [Flo06]. However, another study opposes to this idea by stating that pair programming should be confined to co-located teams; otherwise, it only adds more communication overhead in a distributed context [SS08].

2.4.4 Onboarding

It is not straightforward how to onboard the new team members to the agile methodology practices. Some practices like test-driven development are learned best when someone explains and shows how to do it in person [Mil08]. This gets even more complicated when cultural differences are introduced by distributing the development into multiple countries [SD10].

In this section, some of the disadvantages of applying agile methodology to distributed software development are presented. They include less documentation, meetings times in different time zone, pair programming, and onboarding.

2.5 Summary of the Benefits and the Challenges

This section provides a list to summarize the benefits and the challenges of applying agile methodology in distributed software development as per reviewed literature. Table 1 lists these benefits and challenges.

Benefits	Challenges
Improved Communication and Collaboration	Limited Documentation
Improved Teamness Feeling	Meeting Times in Different Time Zones
More Visibility for the Project Status	Lack of Pair Programming
Continuous Integration of New Code	Challenging Onboarding Process
Faster Time to Market and Responsiveness	

Table 1: List of the benefits and challenges of applying agile methodology in distributed software development

This section determined the following information through reviewing literature: the challenges and the benefits of distributed software development, agile methodology and its two methods called Scrum and XP with their practices, large-scale agile, and finally the benefits and the challenges of applying the agile methodology in distributed software development. In the following section, the research design and methodology of this thesis are presented.

3 Research Design and Methodology

The overall aim of the empirical part of the thesis is to compare the theoretical findings with the results obtained by examining a case. This section presents the study context, the research strategy, and the data collection methods used.

3.1 Study Context

The empirical study was done in one of the international software companies. One specific project team is chosen where there are two central locations, Germany and Poland, primarily taking part, but there are more people from different locations like the US who are helping out to the team, as shown in Figure 1. Scrum and Extreme Programming are the methodologies being used. The project is creating a cloud-native software targeting enterprise customers. The main technologies are the Go programming language and Kubernetes. The role of the author is a member of the team as a software engineer.

Although the author has joined the project one year ago, the project has been continuing for around one and a half years so far targeting to produce a deliverable before a big company event and have all the features needed to be able to offer it within the company software portfolio. The timeline of the important phases is shown in Figure 2. The team has been able to iteratively produce a deliverable per month from the beginning each aiming small but working features. Because of the successful incremental deliverables, the team has been perceived as highly successful within the company.

The project team consists of eight teams, responsible for different components of the product, distributed into two European countries. However, many people are also actively involved in the project (e.g., software architects) in different locations like the United States, which brings the time zone aspect into the picture as well.

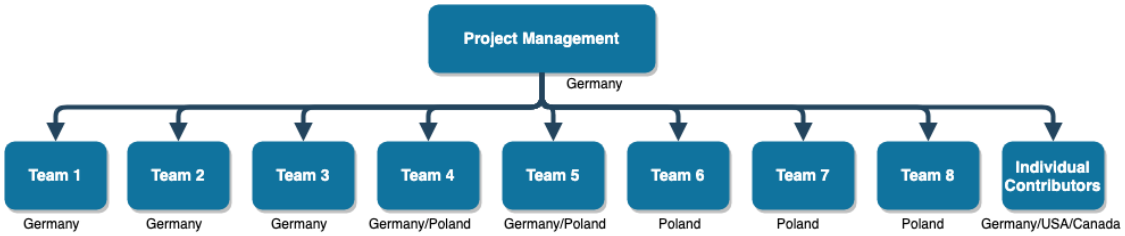

Figure 1: Structure and locations of the project teams.

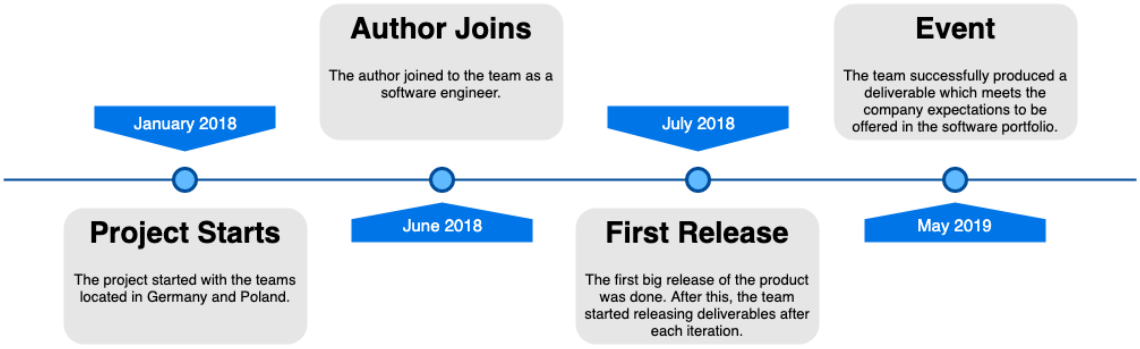

Figure 2: Important phases of the project.

3.2 Research Strategy

A research strategy is determined based on the nature of the problem the researcher is trying to find a solution to. There are mainly two types of research strategies, namely explorative and evolutionary [RB10].

If the researcher is working on a relatively new area to explore more details about the area or to see if it has a potential for further research, then this will be an explorative study. On the other hand, an evolutionary study is where the researcher is working on something already done before and trying to collect different results and come up with a composite answer to the problem under investigation [RB10].

Because of the nature of my research question, evolutionary is the most appropriate strategy. This is because agile methodology in distributed software development has already been in place for years and become a topic for many previous studies. However, the dynamics of the industry is changing, and it is essential to see if agile methodology, in general, still fits in distributed software development.w

3.3 Data Collection Methods

In an empirical study, the data collected can be qualitative or quantitative. Quantitative data require statistics to be analyzed because they primarily consist of numbers. However, qualitative data mostly include words, and that is why they require different techniques like categorization to analyze. Since qualitative data bring forth more details about the subject, most case studies are inclined to utilize qualitative data [Sea99] [RH08].

For this thesis, it is essential to go deeper into details about the subject by examining

different people's point of views. Also, it is directly related to human behavior (e.g., communication, teamness feeling) which is a complex phenomenon that can be studied best with qualitative methods [Sea99]. That is why the approach of this thesis is qualitative.

There are several data collection methods in use for collecting qualitative data, some of which are interviews, analyzing the existing materials like a literature study, and some observation techniques [RH08]. In this thesis, the two methods were mainly used, interviews and participant observation. The author observed the project by being directly involved as a software engineer. Furthermore, the interviews were held with six people from different positions and different teams within the project to compose the empirical part of the study. The respective positions and the teams are shown in table 3.

3.3.1 Participant-Observation

Participant-observation is a particular type of observation techniques where the observer is actively involved in the case studied. It brings invaluable potential to observe the events as an internal so that the observer can make the judgment for the validity of the data collected. Participant-observation also allows examining a case which would not be possible in any other way because of the closeness of some environments like private companies to an external researcher [Yin03].

Also, the author took notes of the events that could be valuable for the study. For example, when there was a misunderstanding between the people working in different locations or when the people expressed their feelings after a daily Scrum meeting with the people from other locations. These were used to remember what had happened and how the people felt.

However, participant-observation can also be challenging for both the participant and the research. It requires the participant's constant attention and makes taking notes or evaluating the events from different perspectives challenging. Also, the participant can take advocacy roles which may not be in the interest of the research. Moreover, the participant can act biased keeping the research in mind and manipulate the events accordingly [Yin03].

In order to reduce the risk of biased research, there is more than one source of evidence in this thesis, which are the notes taken by the author, participant-observation and the interviews. In this way, the case can be evaluated from different perspectives

rather than only from the perspective of the participant, and the triangulation is complete which is important to improve the accuracy of the study [RH08].

3.3.2 Interviews

Interviews are one of the main techniques to collect qualitative data. There are several types of interviews, which are unstructured, structured, and semi-structured interviews [BBJRSR12] [HA05]. For structured interviews, questions are typically fixed, and close-ended, so it is more suitable for a quantitative approach. Unlike structured interviews, in unstructured interviews questions are mostly created during the interview based on the previous answers. The interviewer determines the topic to discuss but has very few questions in mind, so the interview evolves during the discussion [HA05]. For semi-structured interviews, a mixture of the former two, there are specific themes fixed for every interviewee, but the questions may differ based on the flow of the discussion or the role of the interviewee. The researcher puts more emphasis on the themes than the questions [BBJRSR12]. For this research, the way chosen for the interviews was semi-structured. The interview questions can be found in Appendix 1. The determined themes for the questions are some of the challenges of distributed software development to answer the RQ1 and some of the potential advantages and disadvantages of using agile methodology in distributed software development to answer the RQ2 and the RQ3. In this way, discussions in the interviews were targeted towards understanding whether agile methodology has a positive or negative effect against all these challenges in distributed software development.

In this thesis, the interviewees were selected based on the judgment of the author instead of a random selection. The main reason for this was to select people from different positions and different teams to increase the representation of the organization, but also people who are willing to discuss the questions in detail.

Interviews were done during the project was still in progress but already provided a couple of releases, in order to investigate the subject while positive and negative sides were still in effect.

In this research, all the interviews were recorded for detailed examination.

Data Source	Primary/Secondary
Participant-observation	Primary
Author notes	Primary
Semi-structured interviews	Primary

Table 2: Data sources used in this thesis

4 Empirical Results

This section presents the findings for the empirical study. In the first section, challenges and benefits of distributed software development are identified from the answers of the interviewees. In the following section, findings related to the agile methodology, in general, are presented. In the following two sections, how agile methodology affects, specifically advantages and disadvantages, distributed software development is identified. In the final section, the empirical conclusions are presented.

Interviewee answers are quoted and each one referenced with a shortcode of their role. These shortcodes can be found in table 2. For anonymity, all the names or the references to the company are replaced with either "the person" or "the company".

Role	Shortcode	Team
Software Development Manager	SDM	Team 2
Product Owner / Software Architect	PO	Team 4
Scrum Master	SM	Team 1
Software Developer	SD1, SD2, SD3	Team 4, Team 6, Team 1

Table 3: Shortcode explanations referring to the interviewees

To analyze the interview data, first, all the sound recordings of the interviews were transcripted. Then, all the quotes were organized in a table where they were categorized into different aspects such as communication and teamness feeling, and presented with the respective interviewee information. Finally, the conclusions were drawn based on these quotes, which were also presented in this section. This is a technique for qualitative data analysis, called tabulation [RH08].

4.1 Distributed Software Development

Since the team had been involved in distributed software development, they were well aware of the advantages and disadvantages it brought. After learning about the interviewee backgrounds, we started a brief discussion about what they value in software development in general. This is important to see their perception even before talking about distributed software development and agile methodology. Also, it was a good starting point to make them think and warm up for the discussion.

What is important in software development? This question was asked to the interviewees to understand what they value in software development.

Apart from technical qualities, interviewees stressed communication, team collaboration, and the human factor at work as the most critical aspects of software development.

> "The reason I joined this company because it was fun at work. In software development, the human factor is really important. If you are spending eight hours of the day at work, you should also enjoy it. Without fun, I would quit" [SM]
>
> "To be successful [in software development], you need to understand the vision, the problem that we are trying to solve. When you have the vision, you need to have motivated and skilled people working as a team. Teamwork, team collaboration." [SDM]
>
> "Communication is critical in software development. Design is also really critical. You need to come up with a design that is future proof to solve the problem." [SD2]
>
> "Quality and quality management is the most important part of the software development lifecycle." [SD1]
>
> "Communication is really important, but also I must feel comfortable where I work. I mean I must trust the people that I work with so that I can be myself and feel free to share everything with them. After all, they are whom I talk to during eight hours each day." [SD3]

Then, we moved to the discussion about the benefits of distributed software development that they perceived.

4.1.1 Benefits of Distributed Software Development

Labor Cost: Labor cost, as a benefit of distributed software development, was the most common answer from the interviewees no matter what their position was:

> "Usually, it is money to drive companies to distribute the development. I mean, cheaper labor cost" [SM]
>
> "Labor is also very cheap in some countries compared to the others. This is a big benefit for the companies." [SD2]

> "Of course the cost is the primary benefit. They can hire a lot of people in developing countries rather than just a few here." [SD3]
>
> "It is a company strategy to distribute the development. In our case, first location Poland was introduced because of money, the workforce was cheaper, although the people were skilled. In Germany, we have a lack of skilled people, and they are expensive." [PO]

Reducing the labor cost is a big drive for the companies to distribute the development into different countries.

Access to new talents: Another reason to distribute the development is to be able to reach the new talents from other countries. One of the interviewees quickly came up with this answer:

> "It is easier to find new developers. You have a lot more choices than your country." [SM]

Diversity: Diversity is yet another drive for the companies to distribute the development. Distributing the development across globe benefits from diversity, which plays a role in enriching the product:

> "If you have people from different backgrounds, different cultures, different experience, it is super good. In Germany office, we do not have a problem with this. However, if you look at the teams in Poland, they do not have the diversity, and I do not like this." [SDM]
>
> "When different cultures and people involve in the product, this will definitely enrich the product." [SD1]
>
> "Globalization is a game changer. You get different perspectives, different people thinking in different ways. You definitely learn from that. You definitely learn from different cultures." [SD2]

Interviewees referred to diversity as both an advantage and a disadvantage at the same time. The downsides are presented in the following section of the disadvantages of distributed software development.

Time Coverage: Although the time zone is also listed as a challenge of distributed software development, it can be useful as well. The most important reason is that when the time coverage is high due to the difference between the time zones, development never stops:

> "If you distribute the team into other time zones, development lifecycle continues the whole day, which is a big deal." [SD1]
>
> "When you think about different time zones as well, development never stops. You need to have a process for handing over the tasks, but in the end, the product is one, so the company is benefitting." [SD2]
>
> "Advantage of the time zone difference is that you are working on a task and overnight the task may be finished. For example, you could be waiting for a review and overnight the other person continued and provided a review. On the other hand, half of your day, you do not have a chance to communicate with that person. It is a big downside. I think six hours of overlap is acceptable. After lunch, we still had a lot of time together. You just need to arrange it better." [PO]

Also, companies can benefit from time difference to increase the time coverage of customer support.

> "Time difference is beneficial for customer support as well. Sometimes when the customer asks for support, we can support them all day." [SD1]
>
> "Distributing the development is good for supporting the customers. You have more coverage across the Globe in terms of the time. That is a big benefit." [SDM]
>
> "It is definitely increasing the complexity of working. I would say the benefit of distributing the development is having support in different time zones and different locality so that if there is a project in that location, they can easily travel to the customer." [PO]

Labor cost, access to new talents, diversity, and more time coverage are identified from the interviews as the most common reasons why the companies distribute software development. In the next section, the challenges are covered.

4.1.2 Challenges of Distributed Software Development

This section presents the downsides of distributing software development. This is an important section as it covers the first research question (RQ1).

Communication and Collaboration: Communication and collaboration are two closely-related important challenges for distributed software development. In distributed software development, people are limited to use written messages, voice or video conferences to communicate with each other. It is the common feeling that only writing to the other people is not an effective way to communicate because there can be misunderstandings, especially when people do not speak their native language:

> "In written language, you can just assume a lot of stuff, which is challenging, especially when you are not communicating in your native language. When you see the person, you see the emotions and can tell that the person is not annoyed. It is usually just the way how this is done in their culture." [SM]
>
> "Online communication and collaboration are more difficult than on-site. For all of the meetings to do it online is not like drawing things on the white board. Not as effective as face-to-face meetings. To set up an online ad-hoc meeting, you need to find a time when everyone is available. Communicating through Slack, you need to wait until the other person or someone on the channel to notice it and reply to you." [SDM]
>
> "In person communication, I just go to another table to discuss the problem; you see the expressions. There are also other things when you hear the voice or see the message in the chat, it is really different than when you are sitting face-to-face. Few things get lost through the wire. Even if you see them in the video conference, you lose time and important details. Also, the culture can be different, the way of talking, you may feel someone is aggressive or getting defensive, but it is not. Maybe, it is their way of talking, or they are just having a bad day. Being in the other part of the world, you may not be knowing about it." [SD2]
>
> "Another problem is the language. In multinational companies, English is usually the most commonly spoken language, but it is usually a foreign language for most of the employees. Sometimes you do not

> get the right word. English is a foreign language for most of us. How much someone understands what the others say gets changed from person to person. Also, sometimes the phrases may not sound familiar to the native speakers." [SD2]

Furthermore, when there is a disagreement or a divergence in the opinions, it can be more challenging to fix them in the distributed context than co-located teams:

> "When the people in another location discuss something and reach a decision, it is really a problem if you do not agree with that decision. It can be really challenging to do something about this then." [SD2]
>
> "Especially when you are in a conflict with someone, it is really hard to fix this in the distributed team because of less communication." [SDM]
>
> "Nowadays we are having some problems with the other teams as a price of not being able to travel. Somehow the communication is not happening enough. Opinions and ideas are diverging." [PO]

Communication and Collaboration Tools: Although the tools being used for the communication and collaboration can lead to problems because of the low connection quality they provide, almost all the interviewees stated they were happy with the tools they currently used, which are mostly Slack and Zoom.

> "Tooling can be a problem in distributed development as well." [SM]
>
> "I am in general satisfied with the tools. I think we have too many of them at the moment, but it is not a problem. I would always prefer to use the video. I think it should be mandatory if you have a distributed team. It definitely helps. We encourage people to open the camera in the planning and retro meetings." [SDM]
>
> "Zoom is working fine, I am satisfied. Slack is also good for a small number of people. I think for us developers screen sharing is more important than opening video. Slack works fine for screen sharing. Zoom also works like a trump." [SD2]
>
> "Communication tool is a problem of my lifetime. We always had so many different tools, so many channels are popping up, and you are getting interrupted all the time. If you want to have distributed development, you need to assure that it is possible to communicate. But, Slack nowadays is doing a good job." [PO]

Diversity: The interviewees especially emphasized cultural diversity as it adds up to the problem of communication. When culture comes into the play, people should be more careful about what they say considering what this would mean in the culture of the other person:

> "Cultural difference can be a big problem in distributed development. One should be more conscious. For example, in some cultures, it is totally okay to make rude jokes or to be direct. In some other cultures, it is definitely not okay." [SM]

Culture can impact in other aspects than the communication. For example, one of the interviewees mentioned the problem of quick dismissal of the people in Canada (compared to Germany) and how this affected their work together:

> "Also, there is the cultural aspect. We realized that in Canada, they have hire and fire attitude. People were getting fired and had to leave the next day. This was leading to strange situations. You talk to them, and they agree with the decision. Everything looks fine. But the next day, you get an email where they blame you about something. They were not talking openly. I think they are missing that openness either because they do not trust you enough or they are afraid of getting fired. It was really hard to work with them." [PO]

Time Difference: Working with people from other time zones, especially when the span is so high, people can suffer from problems with communication:

> "The most important problem is communication. When the team is located in multiple time zones, you should schedule the meetings according to certain times, and you need to use video communication tools." [SD1]
>
> "To integrate our module with the other team's modules, we need to communicate with the other team a lot, but they are in another time zone. We have very short common time. This is really challenging." [SD1]

To make this work, there should be proper handovers of the tasks between the teams in different time zones, but this may not be sufficient particularly in urgent cases.

"To work with other people in different time zones, you need to work until late. Sometimes you are not even capable of working from home, so you need to come back to the office in the night if there is an urgent problem. Also, sometimes you are solving a problem that is really critical, and then other people just come to the office, and you need to onboard them about the problem" [SD2]

"Some details get lost in the process. Sometimes when you are trying to solve a problem for two hours and when they start working, they just see the problem right away and give tips to solve it in 5 minutes. This is just a waste of time for me then." [SD2]

"Being in different time zones slow down the progress of course. Sometimes you find an issue, and the person you need to ask is in another time zone. You are basically blocked until the next day." [SM]

"Especially if you have different time zones, this is more problematic. Scheduling meetings are cumbersome. Someone is already willing to go home, and others are just starting. In the end, you are having a really short time slot to work together which is challenging." [SDM]

Moreover, a long span in the time difference can lead to working in unusual, non-working times:

"In my previous project, because of the time zone difference, we had only three hours overlap with the other location. We were having the daily standup right before bed, whereas the others were still in the bed early morning. After that, the other team continued working and we had 24 hours working." [SM]

"From time to time I need to stay in the office late to have a call with the people from the United States. I feel usually tired during the meeting after a long day and it is not productive." [SD3]

Teamness: Teamness feeling is based on trust within the team. When someone feels she can trust the others, then she already feels a part of the team. When the team is distributed, it is more difficult to establish trust within the team.

"Distributed software development can only work well if the team is composed of the people who can really function as a team together.

This means there is a high level of trust. If there is an issue with the trust, this will not work. For written communication, someone needs to risk that something will be misunderstood. You really need to have a personal connection with each other to make this work. Otherwise, conflicts can arise." [SDM]

"There are times that you feel you are not part of the team especially when you miss out some discussion and you do not know what is going on. Whenever you are involved, you feel okay." [SD2]

"I think trust and communication are the most critical problems in the distributed development. You do not know them; you do not trust them. Especially, when culture also enters the scene." [SM]

"I see myself as a part of the team, I mean more yes than no. It is mostly because of my previous travels to other locations, I know many people in person. Also, being a Scrum Master, I am the contact person for some people. That is why I talk to more people. That is also helping me to feel like a part of the team." [SM]

"It is challenging when you do not know the other person. You do not know how they think. You need to build trust within the team so that they can work more efficiently. To build trust, they need to meet each other in person at least once." [SM]

Traveling to another location to meet up with the people could help with the teamness feeling, but not every company has a budget to offer this.

"Do I feel like a part of the same team with the people from other locations? With some of them yes, with some of them no. I believe we should spend some time in person with them. We try to find the budget to travel to other locations, but it is hard to achieve. I think knowing someone in person is important because it increases the effectiveness of the communication." [SD1]

"Travelling is important in distributed teams. I am not saying traveling all the time, but at least some of the meetings should be held in one location. If you see people, you trust people more. If you cannot travel at all, at least you should have your video on during the online meetings." [SM]

"In my previous project, we had one big planning event once in a while. All the team members were invited to one location. So, you get

to know all the people, and you discuss the work. If there is anything that you would need to communicate with another team, you would just go and ask them. Afterward, you would party with them which is also a big chance to get to know them better." [SM]

"The companies are underestimating the cost it causes. If you want to work with someone together, you need to see them at least once. Just to get an understanding of their gestures, to get a bit of relation. So, you can evaluate them better, and you get a bit of trust. It is all about trust. Just spending one day with them would be so helpful for trust. This is required, but it is a big cost for the company. On the other hand, if you want to distribute the development, making your employees travel into the other location is a price you need to pay to become more successful. The benefit is well-integrated teams." [PO]

"Team building events help. You get to know the people in this way. If possible, it should be one or two months to work together, exchange ideas, when you come back, that really improves. You have seen the person, have worked with him; now you can picture him not just voice or words." [SD2]

"Product has a certain vision. If we have the same vision, this makes us a team already. Having a common goal in mind. Traveling the other locations always helps with embracing this vision altogether." [SDM]

"I cannot imagine working with a team that works far and does not see each other regularly. That is possible for sure, but I do not think it is working. It cannot work well without having a chance to spend some time with them. At least, knowing some of them would help. This, of course, happens by traveling more." [SDM]

Architectural and Technical Problems: Distributing software development can also lead to problems with the architecture of the software and some other technical problems such as the divergence in the tools or the software development style.

"In our project, each team has its continuous integration pipeline and even uses different tools. They release their artifacts. After that, we integrate all these artifacts into one product. I believe this is not a best practice. Common tools and common understanding of the architecture

is the most efficient way. Everyone in different locations should use the same tools and the same conventions. Otherwise, it is not easy to create a shared understanding of software development." [SD1]

"First, we tried to keep the microservices as independent as possible so that distributed teams could work with their own skills. In the end, they were so different that a person could not work on any other component. There were many inconsistencies between the patterns we used. Nowadays, we try to fix that by providing clear guidance for development. For example, we determined that the programming language should be Go, so at least all the developers can read the source code. That is already helping a lot." [PO]

The challenges with the distributed software development are presented in this section as communication and collaboration, tools, diversity, time zones, teamness, architectural and other technical problems.

4.2 Agile Methodology

After discussing the benefits and downsides of working in distributed software development, we continued with agile methodology. In general, the interviewees believed that the adaption to change is, in general, the primary reason for using agile methodology:

"In Agile [referring to Scrum] you never lose much time. If the customer wants to change the details of the implementation, you can easily adapt in a short time. That is the meaning of being agile." [SD1]

"In today's world, we should be welcoming change, every moment we should expect everything can change. That is why agile is the way to go. Adapting to change is the real reason for being agile." [SD2]

"What are the options we have these days? This is the standard right now. Delivering over iterations, more often, to be able to get feedback and react back to the feedback. That is the biggest benefit. If you see that you are doing something that is not what the customer wants, you can always change the direction. It is better and safer to do this early enough rather than after months of development." [SDM]

"To survive in this world, we need to adapt to the changing requirements quickly. Agile methodology helps a lot with that." [SM]

What if the company gives up agile methodology? This question was asked to understand the overall perception of the interviewees against agile methodology in distributed software development. All the interviewees believed that agile methodology was the best software development methodology available right now and it should not be abandoned no matter what. Some of the answers to the question of what they would do in case the company decides to abandon agile methodology show this:

> "I would say come on, what are you doing? Agile satisfies the customers. You deliver a small working, I am emphasizing the working part, piece of product each sprint. At sprint review, we demonstrate the features to the customers and the stakeholders. We can show them we are working and going in the right direction. In the sprint review, customers and other stakeholders may review your product, and they can see the roadmap. In opposite, in the traditional methodologies, we cannot see the problems until the public release. This will cause lots of problems. Agile methodology cannot be given up." [SD1]
>
> "It will not work. I will get mad and probably take on another job. At the end of the day, the features or even the product will be obsolete. You cannot understand what you will need six months later. You will basically lose the game." [SD2]
>
> "I would leave the company. Why would they do that? There is no other choice for me." [SDM]
>
> "I worked with waterfall before and it was horrible. I feel much more like a software engineer with agile rather than feeling like a person who all day tries to fill up some documents no one reads." [SD3]

Project or Team Specifications: We also discussed the project and team specifications that agile methodology can most fit into. Some interviewees think that it can be applied to anything, whereas some others claimed it should not be applied to the projects where the team size is too big or too small, or the security is at utmost importance:

> "I think agile would work for almost anything. It is a one fits all solution." [SD2]
>
> "I think it works for everything. I even heard other departments like

human resources would start using agile methodologies. I think it is a great idea!" [SD3]

"If you have bigger than nine team members or less than three members, agile development should not be applied. If you have a bigger team, agile will decrease the efficiency of the communication because of many meetings. Also, self-organization is one of the most important parts of the agile, and it would not work. If you have less than three members, then they will always communicate with each other anytime they want. There is no need to have many meetings." [SD1]

"For some certain project that should be done with super quality, secure like a space ship, it is better to go with waterfall. You have design documents, architecture documents. Late but secure. Also, for a spaceship project, you do not change much during the time. You do not need an agile project; you do not need to fail fast, fail at all. For some other projects, where requirements are not that clear, then agile methodology brings a lot of value." [SM]

In general, agile methodology, in general, brings the benefit of adapting fast to the changing requirements and it should not be given up not to lose productivity. Although agile methodology can be applied to any software project, one should think about the appropriate team sizes and the potential security problems in very secure projects because of the lack of detailed documentation.

4.3 Distributed Agile Software Development

Applying agile methodology can bring many advantages in distributed software development to address its most common problems such as communication. On the other hand, agile methodology can also cause some problems in distributed development. This section identifies both the benefits and challenges of applying the agile methodology in distributed software development, which also covers the second and the third research questions (RQ2-3).

4.3.1 Benefits of Applying Agile Methodology in Distributed Software Development

The interviewees were highly motivated to talk about the benefits of agile methodology in general; the author tried to yield them to think more about in the distributed

context:

> "The best thing agile brings is failing fast. If you fail fast, you do not lose much money." [SM]

Communication and Collaboration Because of the number of meetings and their different specifications in Scrum, the team members from different locations can find a lot of time to communicate with each other in different aspects. This increases team collaboration:

> "In distributed teams, communication is the most important part, and one of the agile development methodologies Scrum we are applying provides us communicating in a very organized way. Because we apply all the Scrum meetings. These will increase communication and improves the organization of the team." [SD1]
>
> "Standup is really important for a distributed team. In my previous team, I was the only one who works from a different location. I was missing out a lot. Even though I knew what to do, I was missing out spontaneous discussions. Standups are the only time when I feel involved. The retrospective is really important as well. That is where you know you need to try out different things to improve." [SD2]
>
> "Standups help with the communication a lot. Also, frequent product owner/stakeholder feedback is really valuable; it keeps the product usable." [SD2]
>
> "As I already know them (team members from different locations) from the (Scrum) meetings, I do not hesitate to contact them when I need. I just call the other person on Slack and we have a quick discussion to solve the problem." [SD3]
>
> "You need to have an alignment between the teams. Iterations reviews help a lot with that. We quickly summarize, give a demo. Getting an overview is important. If there is something strange, I just talk to them after the review." [PO]
>
> "During the release planning, we identify dependencies for each team. For these dependencies, we create epics, and all the important communication regarding the issues are done in this epic." [SM]

Teamness Because of the increased communication, agile methodology also helps with the teams to build up trust and increase the sense of teamness among the team members in distributed software development. Also, the retrospective meetings provide a chance for the team to talk openly about any problems and this improves the trust.

> "Retrospectives in agile could also be used to build trust. If you do not have any budget to travel, you can use retrospectives; you can use games for team building so that this will help to build trust. Also, one remark, it is easier to start small for building trust. For example, there is a game where everyone says ten facts about them of which one is not true. Then, others are trying to guess. You can do this on Slack. It is fun, and people are interested, trying to guess. People get to know each other better." [SM]
>
> "We got some help from another team in another location regarding the quality topic. We could just outsource the topic altogether to them, but instead, we integrated them to our agile teams. It just worked great. In the end, we were like a part of one big team." [PO]
>
> "It is important to talk to the team at least once a day. In this sense, daily Scrums are really valuable. To feel like you are in the same team, this communication should happen." [PO]
>
> "These meetings (Scrum meetings) are the only time for me to see and talk to the team members in the other locations. This way, I get to know them and it feels good. I really feel I am in the same team and we work together towards a common purpose." [SD3]

Architectural and Technical Benefits Agile methodology can also help with the software architecture as it provides an incremental working model where the architecture can also be built incrementally. This can help with the problems that distributed software development leads to in the software architecture as these problems can be visible and addressed in each iteration.

> "Distributed development affects the architecture. But since we are agile, architecture also grows incrementally in each iteration. If we see a problem, it is easy to change and adapt." [PO]

Furthermore, agile methodology can be beneficial to adapt the other people in different locations to a common software development style:

> "We have coding conventions that everyone in the team must obey. This makes it easier to quickly adapt and write code in any part of the software when needed. This is especially helpful when the responsible team member is another location and I cannot reach them to ask for explanations." [SD3]
>
> "We had a team from another country, which had a different understanding of the software. We tried to adapt them to writing clean code. We struggled to do that. Sometimes different cultures approach software development in a different way. We applied Scrum practices like review meetings to adapt them to our working model and our vision. It was already helping." [SD1]

Visibility of the Project Status As another benefit, visibility of the project status is also improved using agile methodology in distributed software development:

> "It is also good that we have checkpoints [i.e. sprint review meetings] in Agile, it helps for distributed teams to be on the same page and to understand the current status." [SM]
>
> "Agile helps me regularly see what we do and where we are. This is a great motivation to keep up the good work." [SD3]

To sum up, the advantages of applying the agile methodology in distributed software development include improved communication and collaboration, more trust within the team, architectural and technical improvements, and more visible project status.

4.3.2 Challenges of Distributed Agile Software Development

Although agile methodology brings some advantages over the challenges of distributed software development such as improved collaboration, literature presents some challenges observed by applying the agile methodology in the distributed context.

Constant Change One interviewee mentioned the constant change allowed by agile methodology can sometimes be overwhelming:

> "Agile can be stressful because you change gears quite frequently. Sometimes there can be a steep learning curve. For example, you want to use new technologies in the market to stay relevant and learning can be challenging this way. You have to be productive. If you keep changing constantly, it can get difficult." [SD2]

User Stories Another challenge presented by the PO was deciding the level of details to be included in the user stories.

> "I find it challenging to decide how detailed it should be to present a story in planning meetings. The team can feel that they are independent enough with the story, but also the story is not too fuzzy so that they can do it. Level of certainty and details, in contrast, let us give them some freedom." [PO]
>
> Estimating user stories can also be challenging. It can be difficult to make an estimation before working on a task:
>
> "Estimation is also a challenging thing. How to estimate a story, based on what. We tried different things, but still no certain answer. We tried with Fibonacci numbers, t-shirt sizes, or we just estimated always one to assure that stories are small enough. It is related to planning as well. You just need to have a gut feeling at the end." [PO]

Although constant change and working with user stories can bring some challenges, these are not related to the development being distributed.

Long Meeting Times Another highly stressed challenge was the meeting times in Scrum. They can get too long, especially when the team is distributed.

> "We should keep in mind that we should not exceed the time boxes. Otherwise, the meetings take longer, and the efficiency decreases." [SD1]
>
> "If you would plan for a long time, then all of these meetings are getting unbearable. For example, in planning meetings you have to discuss and discuss, it is hard to discuss that long virtually. For this, shorter sprints are beneficial." [SDM]

"Active participation to the agile meetings can be challenging. For example, in the planning, the product owner explains the user story and expects many discussions. But, a lot of team members can be silent. It is much easier to lose focus in such online meetings. It is really important to be interactive so that all the members can be actively involved in the meetings. Same as in the retros." [SDM]

"Size of the team is also very important when the team is distributed. To make sure that everyone understands what you mean in the story is time-consuming. If they cannot remember what we discussed, then they read the story, but it is not written there. This always brings up a discussion. This problem increases with the size of the team. Quality is always better if the team is smaller." [PO]

Although applying Scrum leads to having many meetings which can take a lot of time, the interviewees suggest this can be mitigated by having smaller teams.

Onboarding Onboarding was discussed to see if it was perceived as a challenge by the interviewees. The interviewees stated that the onboarding process was not a challenge brought by agile methodology. They even claimed that agile methodology made the onboarding quite manageable:

"In general, onboarding someone to the distributed team is more challenging, but agile does not make this more difficult." [SDM]

"We schedule sessions to explain to newcomers how our development lifecycle process is proceeding, how we apply agile development in our team. After two sprints, thanks to frequent meetings, newcomers easily understand how we work, that is one of the best things about agile." [SD1]

"It is not easy to create a shared understanding of software development. There should be training, up-to-date onboarding documentation to adapt the newcomers. Agile meetings help them to reach this common understanding quickly." [SD1]

"If you assign a buddy, that will solve the problem of onboarding. There should be one go-to person for the newcomer from the team. Also, there should be a well-documented procedure which can be visited with each hire. Agile helps with onboarding as well. You have short term

> goals and small stories. Best way to learn is to do. You can get a small task and warm up. Also, you will hear the summaries in the standups and the sprint review and retros." [SD2]

Pair Programming We also discussed the pair programming in the interviews to understand if lacking it in distributed software development harms working with agile methodology. However, thanks to the advanced communication tools in use today, though it cannot be as effective as sitting together, pair programming can still be done in the distributed software development:

> "Pair programming is useful especially for onboarding new people to the team. They can understand the whole idea, tools, frameworks we are using. We just call each other in video conferencing and share our screens for pair programming. It is as easy as that." [SD1]
>
> "In distributed development, you can still share your screen, but it is not as effective as sitting in front of one computer. In case of a review, it is easier to talk to a person and explain your point than writing on the ticket." [SM]
>
> "Screen sharing makes it possible to pair program. I know it is not the same, but still better than nothing. It is so important that it should not be abandoned." [SDM]
>
> "You can always share your screen or even your shell. It is not the same, but it can work out." [SD2]
>
> "I think pair programming is not working as it is written in the book. You do not get together and sit in front of one computer and trying to write a complicated for loop. That is not happening anymore. It is better if someone comes with a proposal, then we sit together and talk about how we should do it better. I think it is more effective. Communication tools are good enough to work on a piece of code together. But, shooting over the table would, of course, happen much more often. That opportunity is still missing. In the distributed context, you first need to review and then call the other person. Just writing your review there is not good enough if there are findings." [PO]

The interviewees believed that pair programming can and should be done in distributed development even though they believed it was not the same experience as being physically together.

Documentation As another potential challenge, the limited documentation in agile methodology was also discussed. Although the author dug more into the documentation as a problem to understand if it was really the case, the interviewees thought this was not a problem. They stated the user stories were supported with any document which could help to clarify the task, and the user stories are already discussed and clarified in the planning meetings.

> "We have technical writers in our organization. When a story is implemented, the developer provides a documentation input like a draft. The developer explains the main parts of this story. Then, technical writes take over from there. They also take part in our Scrum meetings, and they know the details of the story. They ask questions to the developers and expand this documentation both for customers and for internal communication. Thanks to this documentation, anyone from another location could easily develop new features." [SD1]
>
> "Documentation is very important in distributed teams if a feature is not documented well, it is hard to use it. We have a common documentation workspace, and we always keep it up-to-date within our team. If we implement POCs, we always write documentation explaining how we implement it." [SD1]
>
> "Sometimes when we try to integrate our modules with others, we have problems because of lack of documentation. It is also tricky to find a responsible person from another location to communicate. Agile is not the reason for this problem, but the people's responsibilities." [SD1]
>
> "Weaker requirements do not lead to problems. Our PO is very open to ad-hoc discussions. Even if we do not understand a little part of the story, we just write to him on Slack or call him directly. Even if the requirements are not really detailed, we always discuss these requirements in planning and refinement meetings. If there are not enough requirements to implement this feature as customer or PO wanted, we never estimate them. This will cause trouble later." [SD1]
>
> "User stories must be in a form that a developer can understand what he is supposed to create. It leads to more discussions in the planning meeting which is definitely better than reading the long documents. It is more like even a benefit." [SDM]
>
> "Documentation can only be a problem if you do not do things right. If something is not clear in user stories, that should not be even worked

> on. It would go back to the backlog. The stories that being worked on should be clear for everyone even in distributed teams. If needed, of course, a bigger document would be provided until everything is clear." [SM]
>
> "User stories should be self-sufficient. It should not be a problem with distributed development. If we were using waterfall, you would try to understand a 40-page document, where the real feature was starting from page 10. If you start from page 10, something can be missed that could lead to a problem later on." [SD2]
>
> "It is important to have the only relevant documentation. You should not write documents which are not going to be used in the long term. You should have architecture documents where you can both use for the product documentation and for the developers to understand the overall architecture." [PO]
>
> "It should be enough just to have the necessary information for the developer within the ticket." [PO]

Even if there was a misunderstanding and something was implemented wrong, this would not be a big deal:

> "In the distributed context, a developer can misunderstand a story and may not find an appropriate time to talk to the PO and implement something different. The good thing about agile, when you implement something different than the customer wanted, this can be seen in the sprint review. In the worst case, one developer loses one sprint, two weeks. If we implement other traditional methodologies, we would lose more than one month." [SD1]

Although applying the agile methodology in distributed software development presents some challenges such as long meetings times, the empirical study showed that it did not have any more negative effects in other aspects such as onboarding and documentation.

4.4 Empirical Conclusions

Based on the findings from the interviews, the empirical conclusions are drawn as summaries of these findings, which are presented in table 4. Each conclusion is

numbered as EC (Empirical Conclusion) to be used in the next section where these conclusions are compared to the reviewed literature. In total, there are eleven points identified.

Shortcode	Related Section	Empirical Conclusion
EC1	Section 4.1	Distributed software development is preferable by the companies especially to benefit from the lower labor cost in some of the countries.
EC2	Section 4.1	Communication is perceived as the biggest problem for distributed software development by the team members.
EC3	Section 4.1	Even though time zone differences can be beneficial for the development to continue 24 hours, they do harm the team communication and collaboration.
EC4	Section 4.1	Teamness feeling based on the sense of trust is very important for the team members to work productively and collaborate more. It suffers from the geographical distribution of the teams.
EC5	Section 4.2	Agile methodology is well-embraced by the team members, so it plays a role in improving the overall motivation within the team.
EC6	Section 4.3.1	Because of the practices it brings, agile methodology increases the communication and collaboration within the distributed software development.
EC7	Section 4.3.1	Although agile methodology helps to increase trust within the team, traveling to other locations to meet team members face-to-face is still very important for the teamness feeling.
EC8	Section 4.3.2	Pair programming can be done through screen sharing thanks to the current advancement of the tools being used, though it is not as effective as in co-location.
EC9	Section 4.3.2	Limited documentation in agile methodology does not hurt within the distributed software development when the user stories are descriptive enough and are supported with explanatory documents and diagrams.
EC10	Section 4.3.2	Agile methodology does not complicate but even helps with onboarding new members to the team thanks to the constant communication it enforces.
EC11	Section 4.3.2	Some agile practices can be challenging for the team members. These include the number of meetings during a sprint, deciding on the depth of details to be provided for the user stories, and estimating the user stories accurately.

Table 4: Empirical conclusions

5 Discussion

This section presents a discussion on the results obtained in this thesis. The discussion takes place based on the implication that the empirical conclusions (EC), identified in Section 4.4, have on the results presented by the reviewed literature.

EC1 stresses the low labor cost as one of the main motivations of distributing the development. This is highly in line with the literature [Mil08] [PHaOH+10].

Previous literature states that lack of communication creates most of the other problems in distributed software development [Par06] [HM03] [Mil08]. EC2 shows that this empirical study is in parallel with them, as the interviewees perceive communication as the biggest problem in distributed development.

Working with different time zones can affect both positively and negatively the distributed software development. In one hand, it leads to continuous development. On the other hand, it decreases the communication and collaboration [OO00] [HFÅC06]. EC3 supports this as the interviewees referred to time zone difference as a benefit but also a challenge.

Building the teamness feeling is being used in literature as one of the challenges in distributed software development [HFÅC06]. EC4 suggests this is a real problem in distributed development and affects productivity and team collaboration negatively.

A study in the reviewed literature states that agile methodology improves the overall team motivation in distributed development [SD10]. EC5 supports this as the interviewees find agile methodology motivating to keep them working. Furthermore, because of releasing deliverables frequently (once a month) at the end of each sprint with no significant bugs, the team was perceived as successful within the company. As observed by the author, this was another factor to improve the motivation and morale of the team resulting in more dedication to the work.

Both literature and empirical results represented by EC6 refer to improved communication and collaboration as one of the most important benefits of applying the agile methodology in distributed software development [PHaOH+10]. As distributing the work plays a negative role in the communication, agile methodology compensate this thanks to its communication-heavy practices.

EC7 claims that agile methodology helps to increase trust in distributed development, but it does not completely solve the problem. Team members should meet face-to-face at least once to build up trust. This is also supported by the reviewed

literature [PHaOH$^+$10] [SD10].

Although some literature claims pair programming should not be done in distributed teams [[SS08]], EC8 claims it can and should be done in the distributed development as it helps in some aspects such as onboarding new members. Another study in the literature also agrees that pair programming is achievable in distributed software development with the help of communication tools [Flo06].

EC9 and EC10 strongly disagree with literature about the challenges of limited documentation and onboarding brought by the agile methodology to distributed software development. Interviewees argued that these are not challenges but even benefits brought by agile methodology. They thought having less documentation increases the speed and does not cause any misunderstanding as the user stories are already discussed in the planning meetings. They also thought Scrum meetings help to onboard new members to the team. These are in contradiction with the previous literature [Mil08] [SD10].

EC11 shows some new challenges for applying the agile methodology in distributed development. They include the number of meetings, depth of details in the user stories and the estimations for the user stories. These are extra challenges that are not covered in the reviewed literature for this thesis.

Table 5 provides a comparison of the findings from the literature review and empirical study as a summary of the discussions above.

	Literature Review	Empirical Results
Benefits	Improved Communication and Collaboration	Empirical results represented by EC6 shows that because of the practices it brings, agile methodology increases the communication and collaboration within the distributed software development.
	Improved Teamness Feeling	EC7 claims that although it does not eliminate the problem, agile methodology improves the trust within the team.
	More Visibility for the Project Status	Empirical results show that visibility of the project status is improved, mostly because of the review meetings at the end of each sprint.
	Continuous Integration of New Code	This was not discussed in the empirical study.
	Faster Time to Market and Responsiveness	Empirical results show that adaptability to the change is one of the key benefits of agile methodology.
Challenges	Limited Documentation	EC9 shows that, in practice, limited documentation in agile methodology is not a challenge when the user stories are clear enough for the developers and supported by explanatory documents and diagrams.
	Meeting Times in Different Time Zones	Although EC11 points out that the long Scrum meetings can be challenging in distributed software development, the effect of different time zones for this was not discussed.
	Lack of Pair Programming	As per EC8, pair programming can still be done thanks to the advanced screen sharing technologies, but it is not as effective as co-located pair programming.
	Challenging Onboarding Process	EC10 claims that onboarding is not a challenge brought by agile methodology, and it even makes onboarding easier because of the increased communication.
		EC11 also states that the depth of the details in the user stories and the estimations for them can be a challenge as well.

Table 5: Comparison of the reviewed literature and empirical results

5.1 Answer to Research Problem

The research problem of the thesis is: How does agile methodology fit in to overcome the challenges of globally distributed software development? To answer this question, data was gathered from literature and was analyzed versus empirical data. This section presents the answer to the research problem.

Empirical data shows that although globally distributed software development can be preferable by the companies, especially because of low labor cost, it brings many challenges, primarily less communication and collaboration, and decrease in teamness feeling. These are affected even more negatively when there is a big time difference between the locations.

On the other hand, applying the agile methodology in distributed software development reduces the adverse effects of distributing the development according to empirical evidence. It increases communication and collaboration, improves the motivation, and helps to build up trust within the team.

Furthermore, empirical evidence shows that pair programming can still be done in the distributed development and the limited documentation in agile methodology does not do any harm within the distributed setting. Agile methodology is even perceived to be helpful with the onboarding process for the newcomers.

Finally, although there are some challenges brought by agile methodology observed from empirical data such as the number of meetings and depth of the details in the user stories, these are not perceived as blockers from the team and they did not give up working with agile methodology no matter what.

5.2 Limitations

There can be some limitations on the validity of the research done in this thesis. To analyze the limitations, four aspects of validity by [RH08] are used, which are construct validity, internal validity, external validity, and reliability. First of all, although the author asked questions to clarify each answer of the interviewees and also repeated the questions in different ways to make sure the understanding is the same, there can still be different interpretations, which can be a risk to construct validity.

Also, the author was a part of the team that is researched as the case; that is why there is a risk for the internal validity that the results can be biased. However, to

reduce this risk, another data collection method, interviews, is used as well which supports the observation findings. Furthermore, without being a part of the team, this research would not be possible as the company is strict about the externals observing the work being done.

Another limitation is that the empirical study was based on one department of the company. This can be a risk for external validity. Although the interviewees are chosen from different teams and different positions to increase the representability, they still cannot represent the whole organization. There should be more research in different companies and possibly different locations to test the validity of these empirical results.

Finally, the distributed project team was perceived successful within the company, so this can cause a threat to reliability. There should be more research with different teams with failures to increase the representativeness and reliability of these empirical results.

6 Conclusions

This thesis has investigated the suitability of agile methodology in distributed software development. Section 2 looked for the answers from literature to the research questions 1-3, which inquiry the challenges of distributed software development, benefits of applying the agile methodology in distributed software development and challenges brought by the agile methodology to distributed development. In Section 3, the study context, research strategy, and the data collection methods have been presented. Then, Section 4 demonstrated the empirical results obtained by this thesis. Section 5 discussed the empirical conclusions and its implications on the results from the literature. Section 5 also answered the research problem and presented the limitations of the thesis.

Finally, this section concludes the thesis. It first provides a summary of the thesis. Then, the next section identifies the areas for further research.

This thesis goes through the literature and analyzes a case study to find an answer for the research problem: How does agile methodology fit in to overcome the challenges of globally distributed software development? Through interviewing with people and observing the case as a participant, the thesis determines empirical results. According to these results, distributed software development has many challenges, and agile methodology can help the distributed teams, mainly because it increases the communication and collaboration, trust within the team, and the project visibility. This was parallel with the reviewed literature. Some of the reviewed literature claim that some aspects such as onboarding and limited documentation can be a challenge brought by the agile methodology to distributed software development. However, empirical results show that these are not perceived as a challenge, but they present some other challenges such as the number of meetings in Scrum and estimating the user stories.

6.1 Further Study Suggested

There are still some topics that require further research. Firstly, the empirical results identified in this thesis should be tested with additional study. These results are limited to one distributed project team which is perceived successful within the company. Therefore, the same research can be repeated with other teams, maybe even right after some failed projects to see if the results will be different.

Moreover, the research done by this thesis was only in software development, even though the same research can be done in any other fields. It can be interesting to see how the results will look like in the other fields.

References

agi01 Manifesto for agile software development, 2001. URL `https://agilemanifesto.org/`.

AWSR03 Abrahamsson, P., Warsta, J., Siponen, M. T. and Ronkainen, J., New directions on agile methods: a comparative analysis. *25th International Conference on Software Engineering, 2003. Proceedings.*, May 2003, pages 244–254.

BA04 Beck, K. and Andres, C., *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

BBJRSR12 B. Brooks, F., J. Rubin, H. and S. Rubin, I., Qualitative interviewing: The art of hearing data. *The Modern Language Journal*, 80, page 555.

CEP09 Cummings, J. N., Espinosa, J. A. and Pickering, C. K., Crossing spatial and temporal boundaries in globally distributed projects: A relational model of coordination delay. *Information Systems Research*, 20,3(2009), pages 420–439. URL `http://www.jstor.org/stable/23015473`.

Coc02 Cockburn, A., *Agile Software Development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

con Conway's law. URL `https://en.wikipedia.org/wiki/Conway%27s_law`.

CWP08 Cristal, M., Wildt, D. and Prikladnicki, R., Usage of scrum practices within a global company. *2008 IEEE International Conference on Global Software Engineering*, Aug 2008, pages 222–226.

EN01 Ebert, C. and Neve, P. D., Surviving global software development. *IEEE Software*, 18,2(2001), pages 62–69.

ESKH07 Espinosa, J. A., Slaughter, S. A., Kraut, R. E. and Herbsleb, J. D., Familiarity, complexity, and team performance in geographically distributed software development. *Organization Science*, 18,4(2007), pages 613–630. URL `http://dx.doi.org/10.1287/orsc.1070.0297`.

Flo06 Flor, N. V., Globally distributed software development and pair programming. *Commun. ACM*, 49,10(2006), pages 57–58. URL `http://doi.acm.org/10.1145/1164394.1164421`.

HA05 Hove, S. E. and Anda, B., Experiences from conducting semi-structured interviews in empirical software engineering research. *11th IEEE International Software Metrics Symposium (METRICS'05)*, Sep. 2005, pages 10 pp.–23.

HCAF06 Holmstrom, H., Conchuir, E. O., Agerfalk, P. J. and Fitzgerald, B., Global software development challenges: A case study on temporal, geographical and socio-cultural distance. *2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*, Oct 2006, pages 3–11.

Her07 Herbsleb, J. D., Global software engineering: The future of socio-technical coordination. *Future of Software Engineering (FOSE '07)*, May 2007, pages 188–198.

HFÅC06 Holmström, H., Fitzgerald, B., Ågerfalk, P. J. and Conchúir, E. Ó., Agile practices reduce distance in global software development. *Information Systems Management*, 23,3(2006), pages 7–18. URL `https://doi.org/10.1201/1078.10580530/46108.23.3.20060601/93703.2`.

HM03 Herbsleb, J. D. and Mockus, A., An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29,6(2003), pages 481–494.

KA07 Korkala, M. and Abrahamsson, P., Communication in distributed agile development: A case study. *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, EUROMICRO '07, Washington, DC, USA, 2007, IEEE Computer Society, pages 203–210, URL `http://dx.doi.org/10.1109/EUROMICRO.2007.23`.

KB12 Kumar, G. and Bhatia, P., Impact of agile methodology on software development process. *International Journal of Computer Technology and Electronics Engineering (IJCTEE)*, 2, pages 2249–6343.

KRF01 Kobitzsch, W., Rombach, D. and Feldmann, R. L., Outsourcing in india. *IEEE Softw.*, 18,2(2001), pages 78–86. URL `https://doi.org/10.1109/52.914751`.

MH09 Malik Hneif, S. H. O., Review of agile methodologies in software development. *International Journal of Research and Reviews in Applied Sciences*, 1,1(2009).

Mil08 Miller, A., Distributed agile development at microsoft patterns & practices. Technical Report, Microsoft, oct 2008.

OO00 Olson, G. M. and Olson, J. S., Distance matters. *Hum.-Comput. Interact.*, 15,2(2000), pages 139–178. URL `http://dx.doi.org/10.1207/S15327051HCI1523_4`.

Par06 Parnas, D., Agile methods and gsd: the wrong solution to an old but real problem. *Communications of the ACM*, 49,10(2006), pages 29–30.

PDJ09 Phalnikar, R., Deshpande, V. S. and Joshi, S. D., Applying agile principles for distributed software development. *Proceedings of the 2009 International Conference on Advanced Computer Control*, ICACC '09, Washington, DC, USA, 2009, IEEE Computer Society, pages 535–539, URL `https://doi.org/10.1109/ICACC.2009.93`.

PHaOH+10 Paasivaara, M., Hiort af Ornäs, N., Hynninen, P., Lassenius, C., Niinimäki, T. and Piri, A., Practical guide to managing distributed software development projects. Technical Report C12, Aalto University School of Science and Technology Department of Computer Science and Engineering Software Business and Engineering Institute, 2010.

PP11 Pham, A. and Pham, P.-V., *Scrum in Action*. Course Technology Press, Boston, MA, United States, first edition, 2011.

RB07 Ramasubbu, N. and Balan, R. K., Globally distributed software development project performance: An empirical analysis. *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, New York, NY, USA, 2007, ACM, pages 125–134, URL `http://doi.acm.org/10.1145/1287624.1287643`.

RB10 R. Babbie, E., The practice of social research.

RCMX06 Ramesh, B., Cao, L., Mohan, K. and Xu, P., Can distributed software development be agile? *Commun. ACM*, 49,10(2006), pages 41–46. URL `http://doi.acm.org/10.1145/1164394.1164418`.

RH08 Runeson, P. and Höst, M., Guidelines for conducting and reporting case study research in software engineering. *Empirical Software En-*

gineering, 14,2(2008), page 131. URL https://doi.org/10.1007/s10664-008-9102-8.

RMN16 Rolland, K. H., Mikkelsen, V. and Næss, A., Tailoring agile in the large: Experience and reflections from a large-scale agile software development project. *Agile Processes, in Software Engineering, and Extreme Programming*, Sharp, H. and Hall, T., editors, Cham, 2016, Springer International Publishing, pages 244–251.

Sch04 Schwaber, K., *Agile Project Management With Scrum*. Microsoft Press, Redmond, WA, USA, 2004.

SCS06 Sengupta, B., Chandra, S. and Sinha, V., A research agenda for distributed software development. *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, New York, NY, USA, 2006, ACM, pages 731–740, URL http://doi.acm.org/10.1145/1134285.1134402.

SD10 Shrivastava, S. V. and Date, H., Distributed agile software development: A review. *CoRR*, abs/1006.1955. URL http://arxiv.org/abs/1006.1955.

Sea99 Seaman, C. B., Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25,4(1999), pages 557–572.

Sim06 Simons, M., Global software development: a hard problem requiring a host of solutions. *Communications of the ACM*, 49,10(2006), pages 32–33.

Sin08 Singh, M., U-scrum: An agile methodology for promoting usability. *Agile 2008 Conference*, Aug 2008, pages 555–560.

SS08 Sureshchandra, K. and Shrinivasavadhani, J., Adopting agile in distributed development. *2008 IEEE International Conference on Global Software Engineering*, Aug 2008, pages 217–221.

SS17 Schwaber, K. and Sutherland, J., The scrum guide. URL https://www.scrumguides.org/.

ver18 The 12th annual state of agile report', 2018. URL https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report.

VM08 Vax, M. and Michaud, S., Distributed agile: Growing a practice together. *Agile 2008 Conference*, Aug 2008, pages 310–314.

Woh14 Wohlin, C., Guidelines for snowballing in systematic literature studies and a replication in software engineering. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014, ACM, pages 38:1–38:10, URL http://doi.acm.org/10.1145/2601248.2601268.

Yin03 Yin, R. K., *Case Study Research: Design and Methods*. Sage Publications, Inc, third edition, 2003.

Appendix 1. Interview Questions

Background

Could you tell me a bit about your educational and work background?

Why did you join this project?

What is important for you in software development?

Project and Team

What is the overall purpose or project that this agile team is working for?

What is your role in the agile team?

What is the size of the team? How many team members are located in different places?

According to which criteria are the teams built? Are all team members dedicated to this specific project or do they have any other responsibilities?

Are there different roles such as architects, business analyst, or any other individuals called a team member to achieve the defined aim? Except for team members, what other roles are there like the product owner and Scrum master?

Theme: Distributed Development

Do you know the reasons for distributing the development? Are there some particular goals?

What can be the measured and/or perceived benefits of the distributed development?

What are the most critical problems?

Theme: Agile Methodology

Do you know the main motivations to use agile methodology in distributed development?

If your company decides to stop working agile in distributed projects and go back to traditional software management methodologies, what would be your reaction? Would you defend agile? Which arguments would you use to support/oppose agile?

Can you think of any challenges using agile practices introduce in distributed devel-

opment?

Do you think agile methodologies work for all kinds of projects or should they be used only for specific projects? If so, which project specifications should be taken into account to choose agile as a methodology?

Which Scrum meetings are being held on each Sprint? How do you think they affect your own/team's work?

Theme: Communication

If there are product owners and Scrum masters, how do they sustain the communication in distributed development?

What is the scope of communication with teams in other locations?

What tools do you use for communication and sharing the work progress within the team? Are you satisfied with these tools?

Do you use video for the distributed meetings? Is your company encouraging employees to use video?

Have you ever worked in a team which has members from different time-zones? What practices did you follow to achieve a smooth working model?

Have you ever had a situation that you need to discuss with a member in a different location about an issue, but they are not in the office because of the time difference? What was the problem and how did it occur?

Theme: Teamness

Do you feel as a part of the same team with other members in different locations? Are there any actions taken by your company or by your team to improve the team spirit in this distributed project?

Are there any practices promoted by the company to make team members know each other personally, especially about their cultures?

How often different members of the team travel to other locations?

Theme: Architecture

Does distributed development lead to any technical/architectural problems?

What practices are used in your team to mitigate them?

What factors are taken into account for the architecture of the software?

Theme: Onboarding

How do you onboard new members in the distributed team, especially in terms of agile practices?

Theme: Pair Programming

What do you think about pair programming? How do you think lacking it in distributed development affect software development?

Theme: Documentation

How do you treat documentation? Have you ever had a problem in distributed development because of limited documentation in agile methodology?